**Министерство науки и высшего образования Российской**

**Федерации**



**Федеральное государственное бюджетное образовательное**

**учреждение**

**высшего образования**

**«Московский государственный технический университет**

**имени Н.Э. Баумана**

**(национальный исследовательский университет)»**

**(МГТУ им. Н.Э. Баумана)**



ФАКУЛЬТЕТ «Информатика и системы управления»\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

КАФЕДРА «Программное обеспечение ЭВМ и информационные технологии»

**Лабораторная работа № 3**

**Тема** Построение и программная реализация алгоритма сплайн-интерполяции табличных функций.

**Студент** Малышев И.А.

**Группа** ИУ7-41Б

**Оценка (баллы) \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_**

**Преподаватель** Градов В.М.

Москва.

2021 г

**Цель работы:** Получение навыков владения методами интерполяции таблично заданных функций с помощью кубических сплайнов.

1. **Исходные данные**

1. Таблица функции с количеством узлов N. Задать с помощью формулы y = x2 в диапазоне [0..10] с шагом 1.

2. Значение аргумента x в первом интервале, например, при х=0.5 и в середине таблицы, например, при x= 5.5. Сравнить с точным значением.

1. **Код программы**

using System;

namespace Lab\_03

{

class Spline

{

double[,] table\_func;

double[] c;

int n;

int prev, next;

public double Func(double x) => x \* x;

public Spline(double start, double end, double step = 1.0)

{

n = (int)((end - start) / step) + 1;

table\_func = new double[n, 2];

for (int i = 0; i < n; i++)

{

table\_func[i, 0] = start + step \* i;

table\_func[i, 1] = Func(start + step \* i);

}

}

public double GetValue(double x)

{

for (int i = 0; i < n - 1; i++)

if (table\_func[i, 0] <= x && x <= table\_func[i + 1, 0])

{

prev = i;

next = i + 1;

}

GetC();

double a = table\_func[prev, 1];

double b = (table\_func[next, 1] - table\_func[prev, 1]) / h(next) - h(next) / 3 \* (c[next + 1] + 2 \* c[next]);

double d = (c[next + 1] - c[next]) / (3 \* h(next));

return a + b \* (x - table\_func[prev, 0]) + c[next] \* Math.Pow(x - table\_func[prev, 0], 2) + d \* Math.Pow(x - table\_func[prev, 0], 3);

}

private double h(int i) => table\_func[i, 0] - table\_func[i - 1, 0];

private double f(int i) => 3 \* ((table\_func[i, 1] - table\_func[i - 1, 1]) / h(i) - (table\_func[i - 1, 1] - table\_func[i - 2, 1]) / h(i - 1));

private void GetC()

{

c = new double[n + 1];

double[] xi = new double[n + 1];

double[] eta = new double[n + 1];

c[n] = 0;

xi[2] = 0;

eta[2] = 0;

for (int i = 2; i < n; i++)

{

xi[i + 1] = -h(i) / (h(i - 1) \* xi[i] + 2 \* (h(i - 1) + h(i)));

eta[i + 1] = (f(i) - h(i - 1) \* eta[i]) / (h(i - 1) \* xi[i] + 2 \* (h(i - 1) + h(i)));

}

for (int i = n; i > next; i--)

c[i - 1] = xi[i] \* c[i] + eta[i];

}

}

static class NewtonePolynome

{

static double[,] data;

static int len;

static NewtonePolynome()

{

double start = 0.0, end = 10.0, step = 1.0;

len = (int)((end - start) / step) + 1;

data = new double[len, 2];

for (int i = 0; i < len; i++)

{

data[i, 0] = start + step \* i;

data[i, 1] = (start + step \* i) \* (start + step \* i);

}

}

static double GetDividedDiff(double x0, double xn, double y0, double yn) => (y0 - yn) / (x0 - xn);

static (int start, int end) GetDataInterval(double x, int n)

{

int index = 0;

double min\_diff = Math.Abs(x - data[0, 0]);

for (int i = 1; i < len; i++)

if (min\_diff > Math.Abs(x - data[i, 0]))

{

index = i;

min\_diff = Math.Abs(x - data[i, 0]);

}

(int start, int end) interval = (index, index);

if (n % 2 != 0 && interval.start - 1 >= 0 && interval.end + 1 < len)

{

if (Math.Abs(x - data[interval.start - 1, 0]) < Math.Abs(x - data[interval.end + 1, 0]))

interval.start -= 1;

else

interval.end += 1;

}

interval.start = interval.start - n / 2 < 0 ? interval.start : interval.start - n / 2;

interval.end = interval.end + n / 2 >= len ? interval.end : interval.end + n / 2;

return interval;

}

public static double GetValue(int n, double x)

{

var (start, end) = GetDataInterval(x, n);

int nodes\_count = end - start + 1;

double[,] div\_diff\_table = new double[nodes\_count, n + 2];

for (int i = 0; i < nodes\_count; i++)

{

div\_diff\_table[i, 0] = data[start + i, 0];

div\_diff\_table[i, 1] = data[start + i, 1];

}

for (int j = 2, step = 1; j < n + 2; j++, step++)

for (int i = 0; i < nodes\_count + 1 - j; i++)

div\_diff\_table[i, j] = GetDividedDiff(div\_diff\_table[i, 0], div\_diff\_table[i + step, 0],

div\_diff\_table[i, j - 1], div\_diff\_table[i + 1, j - 1]);

double res = div\_diff\_table[0, 1];

double temp = 1;

for (int i = 0; i < nodes\_count - 1; i++)

{

temp \*= x - div\_diff\_table[i, 0];

res += temp \* div\_diff\_table[0, i + 2];

}

return res;

}

}

class Program

{

static void Main(string[] args)

{

Spline myspline = new Spline(0, 10, 1);

double x;

x = 0.5;

Console.WriteLine("Значение кубического сплайна при x = {0:g2}: {1:g6}", x, myspline.GetValue(x));

x = 5.5;

Console.WriteLine("Значение кубического сплайна при x = {0:g2}: {1:g6}", x, myspline.GetValue(x));

Console.WriteLine();

x = 0.5;

Console.WriteLine("Значение Полинома Ньютона 3-ей степени при x = {0:g2}: {1:g6}", x, NewtonePolynome.GetValue(3, x));

x = 5.5;

Console.WriteLine("Значение Полинома Ньютона 3-ей степени при x = {0:g2}: {1:g6}", x, NewtonePolynome.GetValue(3, x));

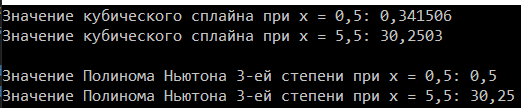
Console.Read();

}

}

}

1. **Результат работы программы**

****

1. **Ответы на вопросы защиты лабораторной работы**
2. Получить выражения для коэффициентов кубического сплайна, построенного на двух точках.

a = y0

b =

d =

c0 = 0, c1 = η1, c2 = 0

1. Выписать все условия для определения коэффициентов сплайна, построенного на 3-х точках.

Ψ(xi-1) = yi-1 = ai

Ψ(xi) = yi = ai + bihi + cihi2 + dihi3

hi = xi – xi-1, 1 <= i < 3

Ψ’(x) = bi+1= bi + 2cihi + 3dihi2

Ψ’’(x) = 2ci + 6dihi =

Ψ’’(x0) = 0, c1 = 0, Ψ’’(x3) = 0, c3 + 3d3h3 = 0

1. Определить начальные значения прогоночных коэффициентов, если принять, что для коэффициентов сплайна справедливо C1=C2.

Т. к. C1 = 0, ξ2 = 0 и η2 = 0, C1 = C2, следовательно ξ3 = 0 и η3 = 0.

1. Написать формулу для определения последнего коэффициента сплайна СN, чтобы можно было выполнить обратный ход метода прогонки, если в качестве граничного условия задано kCN-1+mCN=p, где k, m и p – заданные числа.

CN + 1 = 0

CN = =